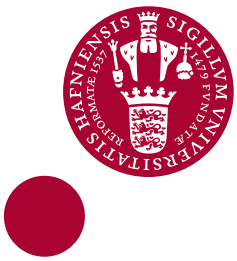

Fast and accurate calculations for cumulative first-passage time distributions in Wiener diffusion models

Blurton, Steven Paul; Kesselmeier, M.; Gondan, Matthias

Published in:
Journal of Mathematical Psychology

DOI:
10.1016/j.jmp.2012.09.002

Publication date:
2012

Document version
Publisher's PDF, also known as Version of record

Online supplementary material

This online supplementary material contains scripts written in R statistical language (R core team, 2012) and Matlab which contain seven helper functions and a main function:

- exp_pnorm(a, b): In $F^s(t)$, large positive a coincide with large negative b in $\exp(a) \cdot \Phi(b)$. This raises numerical issues because values near infinity are multiplied with values near zero. In such cases, we approximated the normal distribution by an exponential (Kiani, Panaretos, Psarakis & Saleem, 2008) such that the product is determined via $\exp(a + b)$ which is numerically feasible.
- K_large(t, v, a, w, epsilon): number of summands needed for the large-time representation of the upper subdistribution $F^\ell(t \mid v, a, w)$. The parameters denote the time, the drift v of the process, the upper barrier a, the relative start point w and the tolerance bound epsilon, respectively. Time t can be a vector.
- K_small(t, v, a, w, epsilon): same for small-time representation $F^s(t \mid v, a, w)$.
- Pu(v, a, w): calculates the probability of absorption at the lower barrier.
- Fl_lower(t, v, a, w, K): calculates the lower subdistribution using the large-time representation $F^\ell(t \mid v, a, w)$. The number of summands is given by K.
- Fs_lower(t, v, a, w, K): same for small-time representation $F^s(t \mid v, a, w)$.
- Fs0_lower(t, a, w, K): same for small-time representation $F^s(t \mid v = 0, a, w)$.
- F_lower(t, v, a, w, sigma2, epsilon): This is the main function for determining the cumulative first-passage time distribution at the lower barrier for a Wiener process with drift mu and variance sigma2 between two absorbing barriers at 0 and a > 0. The function invokes K_small and K_large to determine the number of summands required to attain precision epsilon > 0. The time points t and parameters can be given as vectors. For each element of the vectors t and the parameters, the function automatically selects the representation which requires less terms. Negative drifts and non-unit variances sigma2 are handled.
- F_upper(t, v, a, w, sigma2, epsilon): First-passage time distribution at the upper barrier a.

Example usage

The cumulative first-passage time distributions shown in Figure 1 of the main article have been determined by F_lower(t=1:1000, v=-0.05, a=110, w=80/110, sigma2=3, epsilon=1.5e-8) and F_upper(…) with the same arguments.

R script

```
# Calculates exp(a) * pnorm(b) using an approximation by Kiani et al. (2008)
exp_pnorm = function(a, b)
{
	r = exp(a) * pnorm(b)
	d = is.nan(r) & b < -5.5
	r[d] = 1/sqrt(2) * exp(a - b[d]*b[d]/2) * (0.5641882/b[d]/b[d]/b[d] - 1/b[d]/sqrt(pi))
	r
}

# Number of terms required for large time representation
K_large = function(t, v, a, w, epsilon)
{
	sqrtL1 = sqrt(1/t) * a/pi
	sqrtL2 = sqrt(pmax(1, -2/t*a*a/pi/pi * (log(epsilon*pi*t/2 * (v*v + pi*pi/a/a)) + v*a*w + v*v*t/2)))
	ceiling(pmax(sqrtL1, sqrtL2))
}

# Number of terms required for small time representation
K_small = function(t, v, a, w, epsilon)
{
	if(abs(v) < sqrt(.Machine$double.eps)) # zero drift case
		return(ceiling(pmax(0, w/2 - sqrt(t)/2/a * qnorm(pmax(0, pmin(1, epsilon/(2-2*w)))))))

	if(v > 0) # positive drift
		return(K_small(t, -v, a, w, exp(-2*a*w*v)*epsilon))

	S2 = w - 1 + 1/2/v/a * log(epsilon/2 * (1-exp(2*v*a)))
	S3 = (0.535 * sqrt(2*t) + v*t + a*w)/2/a
	S4 = w/2 - sqrt(t)/2/a * qnorm(pmax(0, pmin(1, epsilon * a / 0.3 / sqrt(2*pi*t) * exp(v*v*t/2 + v*a*w))))
	ceiling(pmax(S2, S3, S4, 0))
}

# Probability for absorption at upper barrier
Pu = function(v, a, w)
{
	e = exp(-2 * v * a * (1-w))
	if(e == Inf)
		return(1)
	if(abs(e - 1) < sqrt(.Machine$double.eps)) # drift near zero or w near 1
		return(1 - w)
	(1 - e) / (exp(2*v*a*w) - e) # standard case
}

# Large time representation of lower subdistribution
Fl_lower = function(t, v, a, w, K)
{
	F = numeric(length(t))
	for(k in K:1)
		F = F - k / (v*v + k*k*pi*pi/a/a) * exp(-v*a*w - 1/2*v*v*t - 1/2*k*k*pi*pi/a/a*t) * sin(pi*k*w)
	Pu(v, a, w) + 2*pi/a/a * F
}
```

```
# Small time representation of the upper subdistribution
Fs_lower = function(t, v, a, w, K)
{
	if(abs(v) < sqrt(.Machine$double.eps)) # zero drift case
		return(Fs0_lower(t, a, w, K))

	S1 = S2 = numeric(length(t))
	sqt = sqrt(t)
	for(k in K:1)
	{
		S1 = S1 + exp_pnorm(2*v*a*k, -sign(v)*(2*a*k+a*w+v*t)/sqt) -
			exp_pnorm(-2*v*a*k - 2*v*a*w, sign(v)*(2*a*k+a*w-v*t)/sqt)
		S2 = S2 + exp_pnorm(-2*v*a*k, sign(v)*(2*a*k-a*w-v*t)/sqt) -
			exp_pnorm(2*v*a*k - 2*v*a*w, -sign(v)*(2*a*k-a*w+v*t)/sqt)
	}
	Pu(v, a, w) + sign(v) * ((pnorm(-sign(v) * (a*w+v*t)/sqt) -
		exp_pnorm(-2*v*a*w, sign(v) * (a*w-v*t)/sqt)) + S1 + S2)
}

# Zero drift version
Fs0_lower = function(t, a, w, K)
{
	F = numeric(length(t))
	for(k in K:0)
		F = F - pnorm((-2*k - 2 + w)*a/sqrt(t)) + pnorm((-2*k - w)*a/sqrt(t))
	2*F
}

# Lower subdistribution
F_lower = function(t, v, a, w, sigma2, epsilon)
{
	a = a / sqrt(sigma2)
	v = v / sqrt(sigma2)
	K_l = K_large(t, v, a, w, epsilon)
	K_s = K_small(t, v, a, w, epsilon)

	F = numeric(length(t))
	i = (K_l < 10*K_s)
	if(any(i)) F[i] = Fl_lower(t[i], v, a, w, max(K_l[i]))
	if(any(!i)) F[!i] = Fs_lower(t[!i], v, a, w, max(K_s[!i]))
	F
}

# Upper subdistribution
F_upper = function(t, v, a, w, sigma2, epsilon)
{
	F_lower(t, -v, a, 1-w, sigma2, epsilon)
}
```

Matlab script

```
% Calculates exp(a) * pnorm(b) using an approximation by Kiani et al. (2008)
function res = exp_pnorm(a, b)
	res = exp(a) .* erfc(-b/sqrt(2))/2;
	d = isnan(res) & b < -5.5;
	if(any(d))
		res(d) = 1 ./ sqrt(2) .* exp(a - b(d) .* b(d) ./ 2) .* (0.5641882 ./ b(d) ./ b(d) ./ b(d) - 1 ./ b(d) / sqrt(pi));
	end
return

% Number of terms required for large time representation
function K = K_large(t, v, a, w, epsilon)
	sqrtL1 = sqrt(1./t) * a / pi;
	sqrtL2 = sqrt(max(1, -2./t*a*a/pi/pi .* (log(epsilon*pi*t/2.* (v*v + pi*pi/a/a)) + v*a*w + v*v*t/2)));
	K = ceil(max(sqrtL1, sqrtL2));
return

% Number of terms required for small time representation
function K = K_small(t, v, a, w, epsilon)
	if((abs(v) < sqrt(eps))) % drift near zero
		K = ceil(max(0, w/2 + sqrt(t/2) / a * erfcinv(2*max(0, min(1,epsilon / (2-2*w))))));
		return
	end
	if(v > 0) % positive drift
		K = K_small(t, -v, a, w, exp(-2*a*w*v)*epsilon);
		return
	end
	S2 = zeros(1, length(t))+w - 1 + 1/2/v/a * log(epsilon/2 * (1-exp(2*v*a)));
	S3 = (0.535 * sqrt(2*t) + v*t + a*w)/2/a;
	S4 = w/2 + sqrt(t/2)/a .* erfcinv(2*max(0, min(1, epsilon * a / 0.3 ./ sqrt(2*pi*t) .* ...
		exp(v*v*t/2 + v*a*w))));
	K = ceil(max(0, max(vertcat(S2, S3, S4))));
return

% Probability for absorption at upper barrier
function P = Pu(v, a, w)
	e = exp(-2*v*a*(1-w));
	if(e == Inf)
		P = 1;
	elseif(abs(e - 1) < sqrt(eps))
		P = 1 - w; % drift near zero or w near 1
	else
		P = (1 - e) / (exp(2*v*a*w) - e); % standard case
	end
return

% Large time representation of lower subdistribution
function Fl = Fl_lower(t, v, a, w, K)
	Fl = zeros(1, length(t));
	for k = K : -1 : 1
		Fl = Fl - (k ./ (v*v + pi*pi*k*k/a/a) * exp(-v*a*w - 1/2*v*v*t - 1/2*pi*pi*k*k/a/a*t) * sin(pi*k*w));
	end
	Fl = Pu(v, a, w) + 2*pi/a/a * Fl;
return
```

```
% Small time representation of the upper subdistribution
function Fl = Fs_lower(t, v, a, w, K)
	if(abs(v) < sqrt(eps))
		Fl = Fs0_lower(t, a, w, K);
		return
	end
	S1 = zeros(1, length(t));
	S2 = zeros(1, length(t));
	sqt = sqrt(t);
	for k = K : -1 : 1
		S1 = S1 + exp_pnorm(2*v*a*k, -sign(v)*(2*a*k+a*w+v*t)./sqt) - ...
			exp_pnorm(-2*v*a*k - 2*v*a*w, sign(v)*(2*a*k+a*w-v*t)./sqt);
		S2 = S2 + exp_pnorm(-2*v*a*k, sign(v)*(2*a*k-a*w-v*t)./sqt) - ...
			exp_pnorm(2*v*a*k - 2*v*a*w, -sign(v)*(2*a*k-a*w+v*t)./sqt);
	end
	Fl = Pu(v, a, w) + sign(v) * ((1/2 * erfc(sign(v) * (a*w+v*t)./sqt/sqrt(2)) - ...
		exp_pnorm(-2*v*a*w, sign(v) * (a*w-v*t)./sqt)) + S1 + S2);
return

% Small time representation of the upper subdistribution (zero drift)
function Fl = Fs0_lower(t, a, w, K)
	Fl = zeros(1, length(t));
	for k = K : -1 : 0
		Fl = Fl - erfc((2*k + 2 - w)*a./sqrt(2*t)) + erfc((2*k + w)*a./sqrt(2*t));
	end
return

% Lower subdistribution
function Fl = F_lower(t, v, a, w, sigma2, epsilon)
	a = a / sqrt(sigma2);
	v = v / sqrt(sigma2);
	K_l = K_large(t, v, a, w, epsilon);
	K_s = K_small(t, v, a, w, epsilon);
	Fl = zeros(1,length(t));
	i = (K_l < 10*K_s);
	if(any(i)); Fl(i) = Fl_lower(t(i), v, a, w, max(K_l(i))); end
	if(any(~i)); Fl(~i) = Fs_lower(t(~i), v, a, w, max(K_s(~i))); end
return

% Upper subdistribution
function Fu = F_upper (t, v, a, w, sigma2, epsilon)
	Fu = F_lower(t, -v, a, 1-w, sigma2, epsilon);
return
```

References

Kiani, M., Panaretos, J., Psarakis, S., & Saleem, M. (2008). Approximations to the Normal distribution function and an extended table for the mean range of the Normal variables. *Journal of the Iranian Statistical Society,* 7, 57–72.

R Core Team (2012). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria.